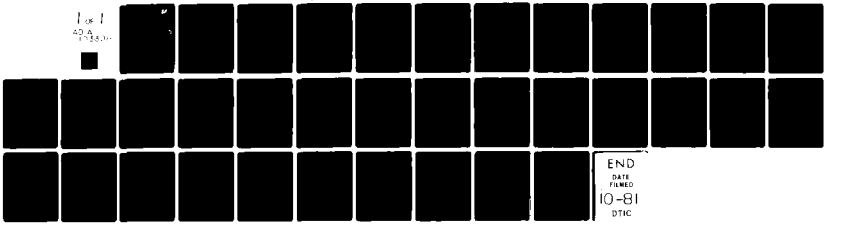
AD-A103 308 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/6 9/2
MDM: HANDLING THE TIME DIMENSION IN GENERALIZED DBMS.(U)
MAY 81 G ARIAV, H L MORGAN MDA903-80-C-0093
UNCLASSIFIED 81-05-06 NL



END
DATE FILMED
10-81
DTIC

LEVEL II

# MDM: Handling The Time Dimension in Generalized DBMS

Gad Ariav and Howard L. Morgan

81-05-06

6 May 81



Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104

This research was supported in part by DARPA under contract MDA903-80-C-0093, and by ONR under contract N00014-75-C-0462 with the University of Pennsylvania.

## ABSTRACT

The sense of time is implicit in almost all human activity, yet it is rarely reflected in the computer database views of these activities. This paper offers a method of dealing with time, modal storage and retrieval, and describes formal and practical realizations of the concept.

The conceptual framework of Modal Data Management reflects three years of experience with the DATA (Dynamic Alerting Transaction Analysis) system [12], in which a time oriented DBMS has been conceived, designed and actually implemented.

The paper first anchors the Modal Data Management concept in the context of the relevant research in Information Modelling and storage technologies, followed by a discussion of the architectural and functional attributes of the DATA system. The major lessons from the DATA experience are then presented, paying special emphasis to their impact on the design of future versions of the system.

The concluding comments deal with specific implications of the Modal Data Management in the domains of DBMS usability and Software Engineering.

## 1.0 INTRODUCTION

We live in a four-dimensional world, yet most computer based systems have limited people to dealing with one or two dimensional objects. The Spatial Data Management System [10], has made an attempt to provide excellent graphical and spatial (3-dimensional) interface to databases. In our real world, however, the fourth dimension, that of time, is equally important. It is our contention that time is not merely another dimension, or another data item tagged along with each tuple, but rather something human users treat in very special ways, since we have so much psychological insight into the meaning of variations with time.

Under the prevailing practices of DB design, largely due to binding constraints on storage and processing capacities, the DB is usually a tenseless collection of "latest available" data. Viewed from the perspective of time, the resulting DB necessarily contains an inconsistent and thin "data-layer". It is important to note that the above design is clearly not a result of the unavailability of temporal data, but rather in spite its abundant availability - one can hardly conceive any form ("transaction") where some temporal data (e.g., date) is not a mandatory item.

Modal Data Management aims at providing the missing link between the deepening recognition of the desirability of time-information in a database and the intensifying prospects of the feasibility of the immense storage and computational capacities which are necessary for that task.

The term "Modal Data Management" is used in this article as a generic reference to the storage, retrieval, and alerting on data, where these functions address directly the inherently dynamic nature of the "real world" which we are trying to represent in Information Systems and their Databases.

"Modal Data Management" intends to invoke the notion of the hierarchy of variability in formal logic, and use the conceptual analogy between Formal Logic and DBMS for clarifying the key idea behind the MDM. In this analogy, the Propositional Logic is the counterpart of the early phases of data management, where data was typically "owned" by corresponding programs. Predicate Logic, the next level in the hierarchy, distinguishes the statement from its argument, which is paralleled in the DB domain by the separation between structure and content [9].

Temporal Modality enhances the variability of the formal description and manipulation of predicates by allowing for temporal changes in individual arguments as well as temporal changes in the structure of basic premises and meaning of predicates. Modal Data Management, analogeously, addresses directly the issue of dynamics, thereby extending the current framework of DBs to handle time in an explicit manner.

The "Conceptual Information Model" (CIM) forms the semantic basis for the contents of the Information System. In a recent paper on the development of such conceptual models, [3], "Time" is perceived as one of the most essential concepts, and models that are based on the time perspective "... provide the conceptual Data Base Model [and] the Conceptual Processing Design with better 'understanding' of the application and the requirements..." (p.401).

MDM addresses the very same issue, namely, the preservation of the dynamic in the modeled world, and therefore appears as the "natural" device for capturing the spirit of the time oriented CIM.

Another development in the IS domain is the emergence of the notion of Decision Support Systems (DSS). As has been noted (e.g. [6], [12]) DSS's major contribution is to the understanding of the relevant problems and the impact of possible courses of action. Understanding, however, as opposed to mere knowledge, is intimately related to the notion of time. "Time, which binds the direction of change, is the rational principle which brings meaningful changes out of chaos; it makes the notion of causal sequence and development possible" ([16], p.18, on the Time in Plato's thought). This last observation suggests that Modal Data Management will probably constitute the data management core of an increasing number of Decision Support Systems, when seeking to investigate causal sequences.

On the supply side, current trends in the development of processing (e.g. associative) and storage (e.g. optical disc) capacities brings us nearer to the point where storage of huge amounts of data, like the one conceivably needed for the storage of "complete histories", will be feasible. Copeland [5], when trying to answer the question "what if mass storage were free?", speculates that a write-once, non-erasable, time-stamped Database that unifies the access to current and past data will be the likely result of the changes anticipated in the cost and performance of mass storage devices.

This is exactly the kind of Database that has been identified as the core component of the Modal Data Management, and has been designed and implemented under the name of the "DATA System", for Dynamic Alerting Transaction Analysis System [13]. The rest of this paper is dedicated to the summary of the three years of experience with two generations of the DATA system and the implications of this experience on the third generation of the system, namely, the evolving concept of Modal Data Management.

Section two summarizes the logical features of the DATA System while the following section surveys the history of the implementations, assesses them, and discusses the major lessons from these experiences.

Section Four discusses the significance of the system and the Modal Data Management in DB usability (methods of use, reliability, recovery,...) and related issues of Software Engineering (software development, DB-restructure,...).

## 2.0 TIME IN MDM

A general discussion of the concept of time is clearly beyond the scope of this paper. The subject has interested philosophers, physicists, mathematicians, logicians, psychologists and others. As a result the concept has had many mutations and still invokes substantial confusion and controversy. (For general reference and comprehensive survey see [8], [16]). A basic consensus, however, is that "time" is an inherent and a

universal dimension of any human experience.

In contrast with the consensus, temporal considerations have usually been factored out of the problem domain in general computer and AI applications. Nevertheless, such consideration start to surface, and examples of similar explicitness are [11] in the AI domain (focuses on the temporal understanding of time oriented data), and [1], which deals with the Conceptual Level of DBMS (it focuses on the identification of the necessary formal features for capturing the semantics of time, time dependent information and various time domains).

Since we are committed to an explicit treatment of the time dimension we cannot elude the time-controversy and feel obliged to identify the kind of time we intend to capture in the Modal Data Management.

Reflecting the various approaches to the term, our DB time is:

1. Linear and unidimensional - we refer to a "date line" or "world line" imagery, and the "arrow of time", namely the orientation of events such that they always flow in one direction, from future, to present, to past.

2. Absolute and objective - we use the (discarded) Newtonian assumption that time has constant flow, independent of an observer's perception. We do not attempt to deal with the notion of experiential time. This is, though, one of our major concerns in the display of time oriented data.

3. Non-hierarchical - namely, we do not treat time periods and their interrelationship as far as stored data is concerned.

4. Explicit - recorded events contain an explicit and unified representation of their "time", and fuzzy specifications which are very common in human expressions, are not valid in the current MDM Database.

We are aware of the severe limitation posed by these conventions. In particular, they put limits on the complexity of stored data and, in certain cases, on its usefulness. We nevertheless feel that the structure of our "implemented time" is necessary to make the problem "managable". We need to learn more about capturing time before we unleash the full potential (and problems) of more complicated and realistic time concepts.

## 3.0 THE LOGICAL FEATURES OF THE DATA SYSTEM*

The DATA System supports a time-qualified user view of a data base, namely, it provides not only the (unqualified) most recent view supported by ordinary DBMS, but also the view of the database as it existed at any given point of time.

---

* The DATA System has grown out of ideas contributed by the authors, Rob Gerritsen of IDBS, Prof. Peter Buneman, Keith Kimball and David Siegel. Their help is gratefully acknowledged.

The DATA System is a relational DBMS where the users view of the data differs substantially from the actual structure of data storage. To support the aforementioned multiplicity of views, the DATA System does not maintain the traditional correspondence between "records" and real world entities - it contains rather cumulative, time ordered, lists of transactions. In this way, the status of an entity at any given point of time, say, t, is being extracted from the collection of transactions relating to that entity, which have been recorded prior to time t, while ignoring all the transactions since t.

For example, the current status of an employee ("current" is just a special case where t="now") is derived from his/her "hiring transaction" and all the transactions that have followed as the time passed, namely, change-of-address, pay-raise, job-assignment and the like. Another example is visualized in Figure 1.

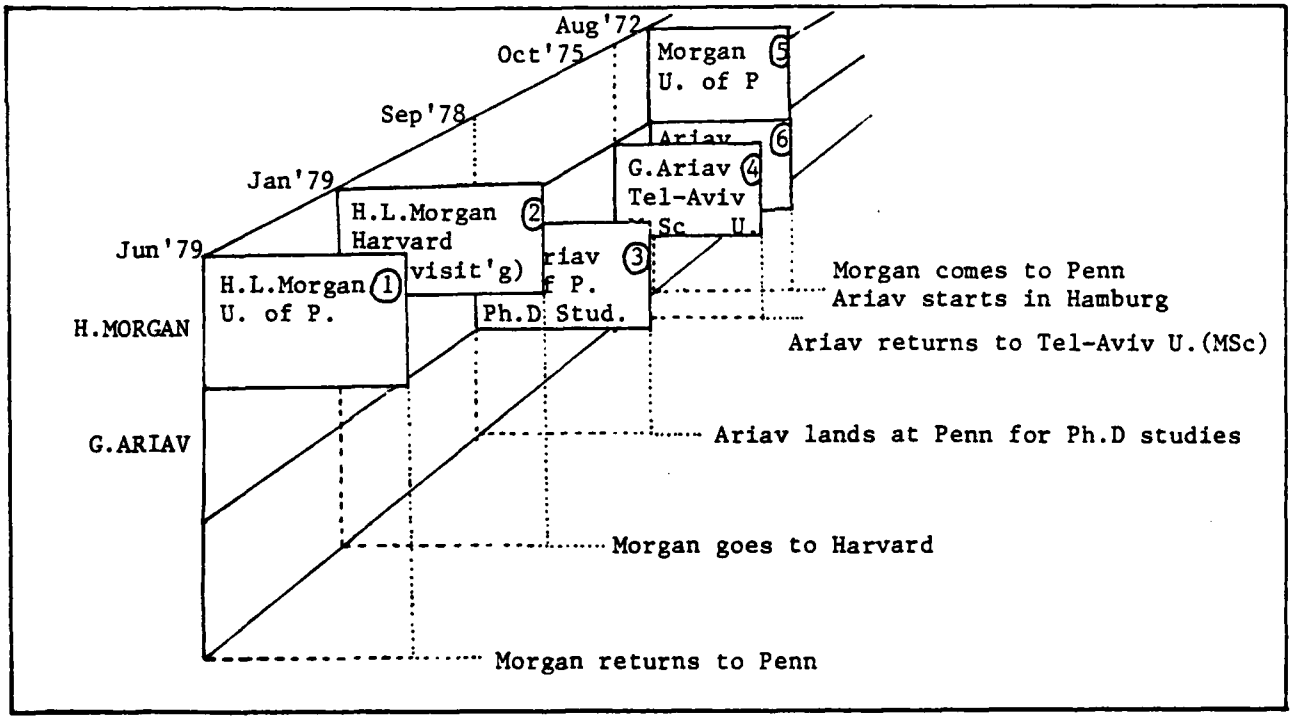



FIGURE 1. The Logical Storage in the DATA System.

Physically, transactions are always appended to the DB, and they are never modified or deleted. Nevertheless, they can be one of three basic types, which reflect the three possible types of events in the modelled world:

1. "Addition" - the introduction of a new instance of an entity,
2. "Modification" - a change in an attribute of an existing instance of an entity, and
3. "Deletion" - the removal of an existing instance of an entity.

Beyond its type, each transaction contains its time stamp, a pointer to the previous transaction related to the same entity and eventually the new value of any data changed by the transaction.

This structure has the composite effect of addition, modification and deletion of "records" in the ordinary DBMS, though the dynamic database is constantly growing.

This underlying structure resembles the human perception and cumulative knowledge of the "world", where, for instance, the firing of an employee removes him from the current list of employees but does not "delete" his association with the organization at a previous point of time.

The architecture of the DATA systems builds on the notion of "Differential Database" [15], where a small data set contains changes to a large (read only) Database. The DATA system actually extends that notion to the point where the background database is being eliminated (it is an

empty set) and its "read only" characteristic now applies to the collection of the "changes".

An overview of the actual implementation history of the DATA system is defered to the next section. In the current discussion we will concentrate on its logical structure.

## 3.1 General Structure

The system includes two basic procedures (see figure 2), namely, the DB/DESCRIBE ("DDL" processor) and the DB/PROGRAM ("DML" Processor).

The DB/DESCRIBE module elicits and maintains the description of the DB. The required elements are the identification of the DB, the various relations in it, and the definition of the various domains in the corresponding relations. These definitions are stored in the predefined relations DATABASE, RELATIONS and DOMAINS, which together constitute the definition of the DB to be maintained by the DATA system.

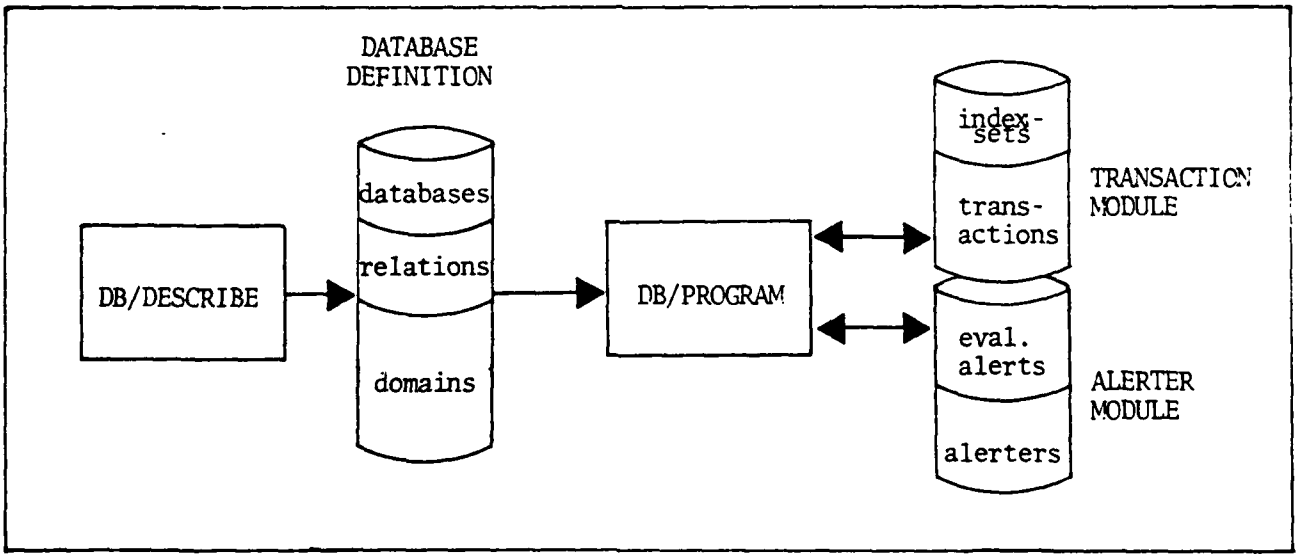



FIGURE 2. Diagrammatic Overview of DATA Systems' Components.

The database is accessed by running a program called DB/PROGRAM. This program manipulates two basic structures. The first structure is called TRANSACTIONS and consists of a time ordered list of transactions. The format of this structure is:

```
TRANSACTIONS RELATION  (TRANS-DBNUMBER KEY NUMBER;
                        TRANS-NUMBER KEY NUMBER;
                        TRANS-TYPE NUMBER;
                        TRANS-RELNUMBER NUMBER;
                        TRANS-PREVNO NUMBER;
                        TRANS-YEAR NUMBER;
                        TRANS-DAY NUMBER;
                        TRANS-HOUR NUMBER;
                        TRANS-MINUTE NUMBER;
                        TRANS-SECOND NUMBER;
                        TRANS-DATA ALPHA 512)
```

The TRANS-DBNUMBER identifies the database affected by the transaction. The TRANS-NUMBER is a number assigned to distinguish betwen transactions on a given database. The TRANS-DBNUMBER and the TRANS-NUMBER identify a transaction. The TRANS-TYPE specifies the type of transaction. The types of transactions are addition, modification and deletion. The TRANS-RELNUMBER is the relation number of the tuple being affected by this transaction. TRANS-PREVNO is the TRANS-NUMBER of the previous transaction on the entity. All transactions upon an entity are linked via this domain. TRANS-YEAR, TRANS-DAY, TRANS-HOUR, TRANS-MINUTE and TRANS-SECOND datetime stamp the transaction. TRANS-DATA is the new tuple. In practice only the number of characters necessary to contain the new tuple are stored in the database.

The other basic structure manipulated by DB/PROGRAM is called INDEX-SETS. The tuples of INDEX-SETS are used to locate the most recent transactions upon the entities. This structure provides the head of the

linked list of transactions upon an entity. The structure is as follows:

```
INDEX-SETQ RELATION    (IS-DBNUMBER KEY NUMBER;
                        IS-RELNUMBER KEY NUMBER;
                        IS-KEY     ALPHA 256;
                        IS-DELETE NUMBER;
                        IS-TRANS-NUMBER NUMBER);
```

The IS-DNUMBER and IS-RELNUMBER identify the database and relation to which the key belongs. IS-KEY is the key uniquely identifying a tuple. IS-DELETE is boolean indicating whether or not there is a current tuple in the database with the key value and then was deleted from the database. IS-TRANS-NUMBER is the TRANS-NUMBER of the last transaction upon the entity. This provides the head of the linked list of transactions upon the entity.

The repertoire of DATA's DML commands includes four sets, summarized in figure 3 and discussed briefly below. In our discussion we concentrate on the specific features of DATA which distinguish it from traditional DBMS.

DATA COMMANDS

I. DATA MAINTENANCE:

ADD
MODIFY
DELETE

II. DATA ACCESS:

SHOW (for perusal of the DB definition)
DISPLAY
RUNDOWN
(chained DISPLAYS for perusal of the "history" of an entity from given point in time and on)
FIND

III. TIME-CONTEXT HANDLING:

SET-TIME
(allows the viewing of the DB at a specified point of time)
ROLLBACK (backs the DB up in time)

IV. ALERTING:

CREATE
DISPLAY-ALERT
ENABLE
DISABLE
CANCEL

FIGURE 3. DATA's DML Comands

## 3.2 DATA MAINTENANCE COMMANDS:

The ADD command creates a new tuple in the database. This command contains a relation name and initial values for various domains. It verifies that no tuple logically exists with the same key value as the tuple being created, and a transaction of type "Addition" is entered on the list of transactions. If there was a previous transaction on the

entity, the corresponding tuple of INDEX-SETS has its IS-DELETE domain reset. This tuple points to the new tuple of TRANSACTIONS.

The MODIFY command requires a tuple to be currently located (via a previous FIND or ADD command). This command contains a relation name and new values for non-key items. The tuple is entered on the list of transactions with the new values of the non-key items. The corresponding INDEX-SETS' tuple is pointed to the new transaction.

The DELETE command deletes logically a tuple from the database. This command contains a relation name and values to which the key must match. First a tuple is located by matching the specified key values. Next a transaction is appended to the list of transactions with a type of deletion. Finally the tuple in INDEX-SETS is marked with IS-DELETE as 'true' and pointed to the recently created transaction.

## 3.3 DATA ACCESS COMMANDS

The FIND command requires a relation name and values to which keys must match. The program searches INDEX-SETS for a tuple matching the specified key values. Upon successful location of the corresponding index tuple, the corresponding transactions for this entity are read until one is found with a datetime stamp less than the time that the database is being viewed from. If such a transaction exists and its type is not a deletion, the command is successful, else an exception is returned stating that there was no such tuple at the specified time.

The DISPLAY command causes values in the current tuple of <relation name> to be retrieved and displayed.

The RUNDOWN command "replays" the sequence of recorded events that relate to a specified tuple between two specified points in time. This command requires a relation name, tuple's key and a specification of the desired time interval. The starting point of the sub list is being located and all the transaction from that point, up to the specified end time, are being retrieved and displayed.

The SHOW command is used to view the definitions in the DB Description. SHOW by itself displays all relation names of the database. SHOW <relation name> displays the domain names of that relation. This is accomplished by reading the RELATIONS and DOMAINS relations.

## 3.4 TIME-CONTEXT HANDLING COMMANDS

The SETTIME command allows the database to be viewed from previous points in time. SETTIME=CURRENT states that the database should be viewed from the current time. SETTIME by itself asks the time that the database is being viewed from. SETTIME= 77/155 @ 23:12:11 implies that the database should be viewed as it existed at 11:12:11 p.m. on the 155th day of 1977.

The ROLLBACK command backs up the database to a specified point in time which allows the user to view and operate on the DB content at that point in time. This is accomplished by simply updating pointers in

INDEX-SETS and then reading the corresponding TRANSACTIONS tuple. If the TRANSACTIONS' tuple is beyond the rollback point, the linked list of transactions on this entity are read until one is found before the rollback point. If a transaction is found before the rollback point, the INDEX-SETS tuple is updated to point at this transaction and the IS-DELETE domain is updated based on the type of transaction. If no transaction is found before the rollback point, the INDEX-SETS tuple is deleted.

## 3.5 ALERTING COMMANDS

An Alerting system provides facilities to monitor changes to the database in order to perform some action whenever certain conditions become true. Alerters are the basis of the alerting system. An alerter associates a name (of the alerter), a condition to be evaluated, and an action to be taken when the condition is met. An alerter can be thought of as a program that continuously monitors the database and takes some specified action when the specified condition becomes true.

Alerting changes the concept of a database management system from a passive to an active system. Previously the user/database interface consisted of a subroutine call. The user would query the database system and then wait for a response. The user program and database system did not operate in parallel and the database responded only when spoken to. Under an active database system, user programs and database systems operate in parallel and both are capable of action. Alerters in this system are restricted to simple alerting conditions [4], and the associated action is merely a display at the terminal. Further discussion of Alerters and the various types of Alerters is included in the section

that highlights the system's features, later on. The creation of an alerter is done by the ALERT command. For example:

```
ALERT BIGRAISE (MODIFY OF PEOPLE) =
                SALARY.NEW > SALARY.OLD * 1.1
```

BIGRAISE is the name of the alerter. The MODIFY OF PEOPLE states that the alerter should be evaluated after the modification of the relation called PEOPLE. The condition is SALARY.NEW > SALARY.OLD*1.1 (i.e. a 10% raise).

The Alert is being stored in a separated type of relations (see Figure 2).

DISPLAY-ALERT <alert name> causes the text and state of <alert name> to be displayed. ALERT <alert name> = CANCEL causes <alert name> to be cancelled. An alerter can be in one of two states. If an alerter is enabled, it is a candidate to be triggered. If an alerter is in a disabled state, the alerter may not be triggered until it is enabled. The alerter may be placed in these states by the commands ALERT <alert name> = ENABLE or by ALERT <alert name> = DISABLE.

## 4.0 ASSESSMENT OF DATA'S IMPLEMENTATIONS

The System has been implemented as a relational DBMS, on top of conventional network (CODASYL) DBMS, and providing a stand-alone DB-access, namely the various commands to access and maintain the DB are not embedded in a host programming language, but could be conceived as a

user-interface of DB-server in a multi-tasking environment.

As has already been mentioned, the system has evolved through two versions of implementation. DATA-I was originally constructed on a Burroughs System, using DMS-II, a CODASYL compatible DBMS, as the underlying data management facility.

The second version, DATA-II, was aimed primarily at improving the system's availability and portability through the use of PASCAL as the implementation language and SEED, an extremely portable CODASYL compatible DBMS [14], as the basic tool for the data handling. As far as the functional structure is concerned, the repertoire of commands was enhanced by the introduction of the RUNDOWN command, to improve data accessibility.

The three segments of the DB (figure 2) are realized as three groups of (CODASYL) records and sets, as depicted in figure 4. Note that DATA-II tracks and monitors the data entry, and records automatically the identity of the person who introduced the transaction into the DB.

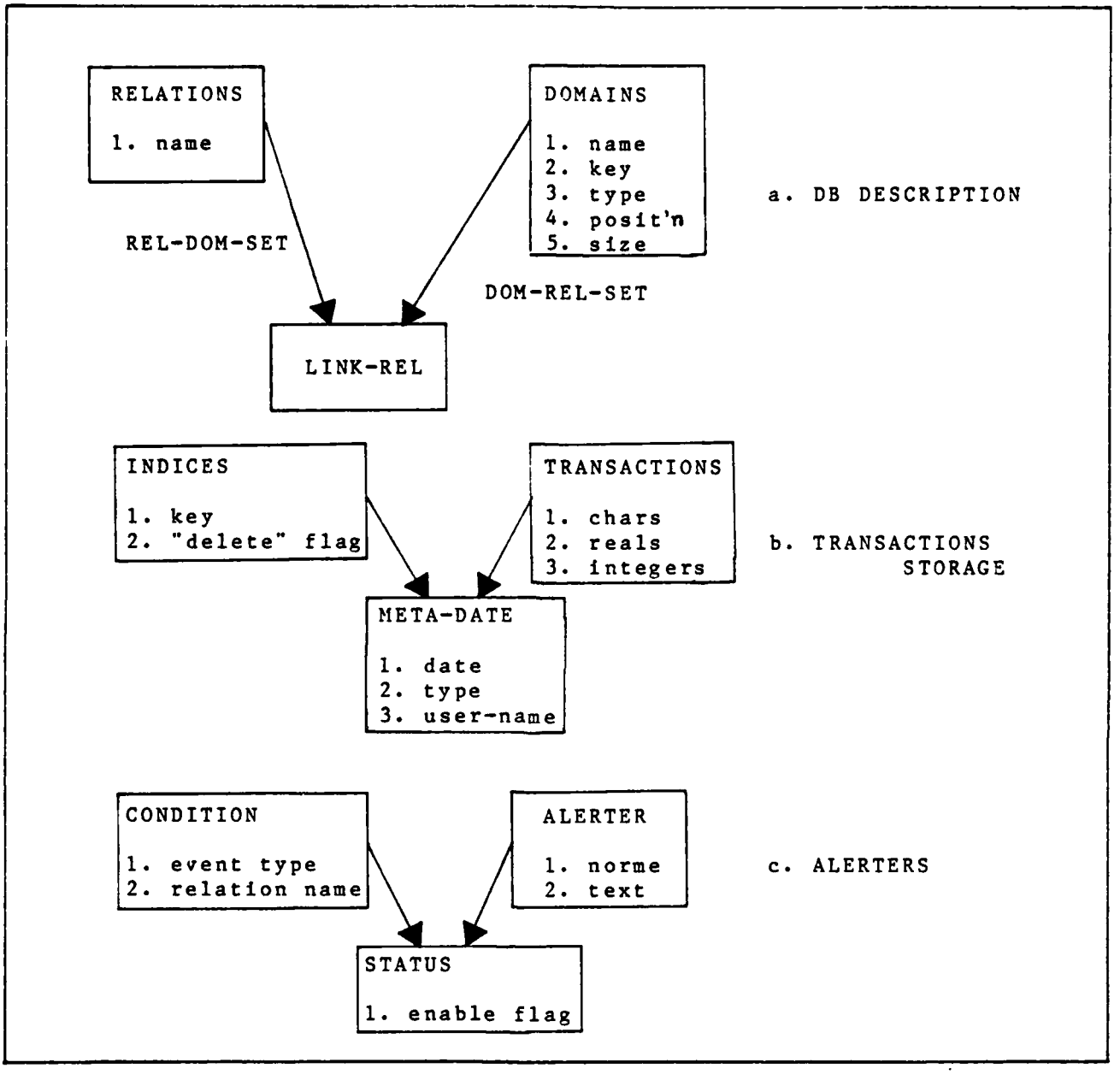



Figure 4. DATA's Data Structures

During the years of experimenting with the two versions of the system we have clearly learned a lot about what we have labeled as Modal Data Management. Following is a summary of our major observations and the resulting recommendations for the upcoming version of the Modal DBMS.

1. The concept has proved to be feasible -- It has been demonstrated in couple of test applications. It was found to be useful and was generally well received.

2. The notion of the recorded time involves some confusion, namely - is it the time of recording or time of occurance? In the latter case, if the DB resides on a write once media, transactions need to be stored in a "Quiscing-Area" first, and moved (periodically) in the correct sequence to the permanent storage area.

3. The stand-alone, user oriented DB access was found inadequate. While the notion of "DB-server" fits our computer network environment (DARPA), the DB-server (as a "central Version") has to function as an interface between the DB and programs that run against it.

4. While the ability to retrieve sequences of events proved to be a powerful feature, the conventional modes of representation and display do not assist in effectively invoking the sense of time in the user/viewer. This observation has ushered in a parallel research effort to explore more powerful methods for the display of time oriented data, [2].

5. The system has emphasized the containment of the dynamics in the data itself. It became, however, apparent that the system has to address the metadynamics, namely, the temporal changes in the schema itself. This observation is even more pronounced in the use of write-once media, and it seems to be the key issue for the temporal unification of access to data. In terms of the future

design, this means that we need to support multiple, time ordered (internal) schemata, through which programs will gain access to the portion of data that has been recorded while the corresponding schema "prevailed". Under the same notion, the SHOW command also needs to have certain time-context. Incidently, this observation has inspired us to select the term Modal Data Management as label of our effort.

6. The relational design/appearance of the DB has been exploited successfully in structuring the DB and the control over the storage and maintenance of data. However, the user does not have any of the more powerful relational operators at his/her disposal, for instance he/she cannot JOIN two relations at a prespecified time context to deduce information from the transient view of the DB.

7. The architectural decision to support only simple alerters as an integral part of the DBMS seems to be justified. The isolation of the complex alerting in a dedicated module simplifies substantially the DB/PROGRAM.

8. In accessing data, some capacity to deal with fuzzy time expressions seems to be necessary, e.g, to operationalize the expression "about six years ago".

9. Supporting the dynamics of the data necessitates the use of internal identification of entities, since we cannot assume -- in the long run -- that the logical keys will remain unchanged.

10. Efficiency consideration - a balance must be struck between the storage of single transactions and duplication of the cumulative status of the entity. Similar concern is the possible distinction between "relatively stable" and "relatively-active" parts of relations, and the response time implication of implementing such a structure.

11. The system supports only questions of the type "what happened at time T?" Where T is a temporal expression (point or an interval). The other necessary time oriented question is "when did event X happen?" where X may be of various levels of complexity.

## 5.0 IMPLICATIONS

The core Database of the Modal Data Management, as has been developed in the previous sections, has some major (potential) implications in the domain of Database Usability and in certain aspects of Software Engineering. Some of the implications, however, affect both concerns. For instance, improved testing conditions will result in more stable systems which will in turn, increase the systems usability. For the sake of clarity, such implications are arbitrarily classified in one category, based on our perception of its primary effect.

Part of the implications stem from the underlying "differential file" approach [15] while another part is due to the unified access to current and past data, (partially speculated or alluded to in [5]), yet some implications are unique to the architecture and concepts embodied in the

system.

## 5.1 Usability

In this domain, the major impact of the Modal Data Management is in the enhanced meaning and usefulness of the DB to set of traditionally ill served users, the support of Complex Alerting, the improved basis for DB integrity, and increased availability. Following is a brief discussion of each.

### 5.1.1 Functional Support

Following is a demonstration of the new capacities provided by the MDM concept; It is clearly just one single example out of many. "Time-Travels" are an essential component in Business Planning in general, and especially in the Retrospective Analysis, namely in the attempt to trace and understand performance deviation. This necessitates the ability both to "freeze" the state of the "knowledge-about-the-world" (i.e., the Database) at the time of the original Planning and to reconstruct that "frozen" view at a later Point in time to answer questions like "what would have been the projected performance if we had known at that time what we know today about the production process".

The effort required for such advanced Decision Support is clearly beyond the capabilities of typical current systems, and believed to be a major reason for the rare attempts to conduct such Retrospective Analyses [7].

"What/Where/Who is ..." are questions of primary importance and justifiably dominated the scene of data management. "What/Where/Who was ..." is a question that people ask also quite often, especially while trying to discern and understand causal relations. Functionally, the Modal Data Management treats these two types of questions symetrically, thereby extending substantially the range of the "What if" questions that the DB can support.

### 5.1.2 Complex Alerting

Previous works in this area [4] have identified three levels in the hierarchy of alerting conditions, starting with "simple", through "structural" and ending with "complex" alerting conditions.

Simple alerting conditions deal only with one relation and the condition can be evaluated by referencing only the old and new copies of the tuple being changed. An example of a simple alerting condition is "tell me when anyone receives a raise over 25%".

Structural alerting conditions deal with changes in the structure of the database. These require the monitoring of more than one relation. An example of a structural alerter is "tell me when a person under 30 years of age owns a car priced over $10,000". This requires that both the CARS and PEOPLE relations be monitored.

Complex alerting deals with a more global view of the database, and it has been broken down into two classes. The first class of complex alerting conditions require the use of time. An example of a time spanning alerting condition is "tell me when a car has three price

increases in one year". The second class of complex alerting conditions are statistical alerts. An example of a statistical alerter is "tell me when the average price of a car is over $5,000".

The Modal Data Management DB clearly enables all the three levels, in particular it supports the complex alerting conditions which require the ability to view the DB at previous points in time, as well as the tracing of the changes and sequences of changes the entities have undergone.

A different aspect with regards to alerting is the unique environment for testing proposed alerters. Through the preservation of the dynamics in the modelled world it allows to repeat sequences of events and simulate, at will, actual past scenarios. The long and not so successful search for a test environment for implemented alerting systems has taught us about the importance of that feature. Beyond that, it supports complex modes of pattern recognition and advanced data analysis.

### 5.1.3 DB Integrity

No information is ever lost due to update when using the MDM database since the physical database is simply a list of transactions, and once a transaction is entered on the list, it is never modified.

The database provides the ability to place consistency constraints that could not be placed on a conventional database. An example of such a constraint is "accept no more than ten changes from a user within one hour". This constraint can be enforced by viewing the database as a series of time-stamped transactions, in which it seems to be difficult to hide nefarious activity.

### 5.1.4 DB Availability

A MDM database can be easily dumped and restored - The list of transactions are simply appended. Moreover, the only part of the files which need to be dumped are those added since the last dump. This substantially reduces the time required to dump the database. Since transactions are simply appended, there is no danger in dumping the database while it is online.

Fast recovery from hard loses is easily provided. If a block on the disk is destroyed, it can be simply reloaded from a backup dump and processing can continue. This is possible because transactions are never modified once they are entered on the list of transactions. The database could be allowed to run while blocks were being copied.

Soft loss situation occurs when programs incorrectly affect the content of the DB. The Modal Data Management DB provides recovery from soft loss without maintaining a log tape. Backing up the database to a previous point in time is accomplished by simply discarding all the transactions that were recorded after the error had occurred.

## 5.2 Software Engineering

The major impact that the concept of MDM has on Software Engineering is in an improved program-development environment and elimination of some design anomalies. The more impressive implication, however, is the elimination of the need for DB-restructuring. This is a result of the basic Modal premise of the system - it internalizes changes in the data and in the structure as well.

### 5.2.1 Program Development

The development of application programs can be simplified by a MDM database. At some desired point in time the list of transactions could be branched into two separate lists (see figure 5). One list would constitute a production system and the other list would constitute a developmental system running in parallel. Both systems would share the same base list of transactions. Any program being tested would update the database via the developmental list of transactions. These changes will not affect the production database. Thus programs can be tested against the large central database without affecting the production database.

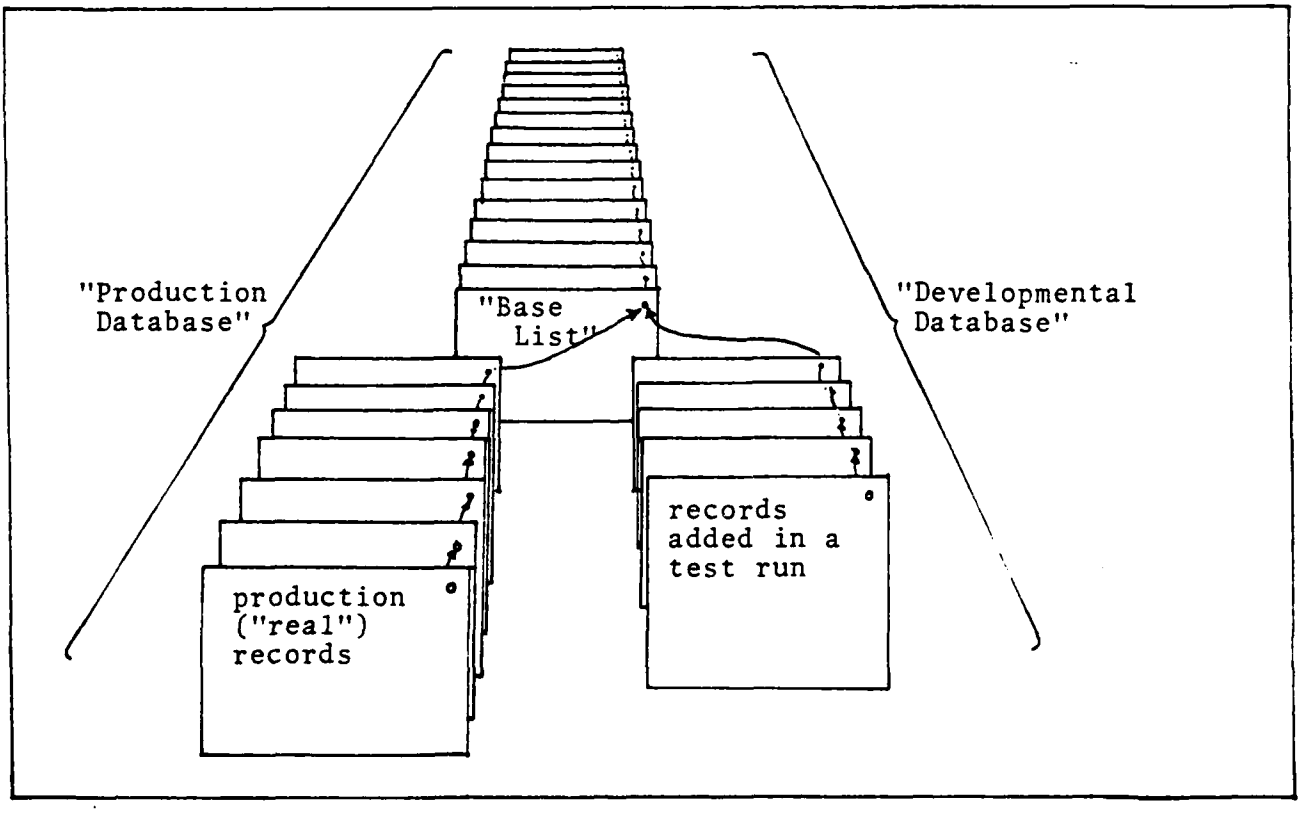



FIGURE 5. Test Data in a "Fork" Mode

This mode of operation overcomes, to a great extent, the chronic problems of test data and the need for realistic operational conditions in the test phase.

### 5.2.2 Mixed Access Anomalies

Some applications are required to be online for long periods of time and still require reports to be generated, e.g, accounting reports as of end of period. This has lead to awkward models in order that the reports may be generated with consistent data. The MDM's DB could discard these awkward models since it provides the ability to freeze a program's view of the database. Thus online updating of the database could continue while the report is being generated.

### 5.2.3 Elimination Of DB-restructuring

We use the term restructure to indicate changes in the logical structure of the DB, and differentiate it from "reformatting" - changes to the physical structure. Both constitutes DB-Reorganization, as it is being defined in [18]. Usually all data bases need sometime to be restructured, depending on the rate of the relative decay in the functional capabilities of the system as time passes and the user environment evolves and changes.

Common instance where some restructuring is inevitable are [18]: internal organizational change (e.g., workers are assigned now to two projects instead of one), external organizational change (e.g., merger, where customer bases are consolidated) or legislation (e.g., dropping data

items not necessarily required).

DB-Restructuring is usually a very complicated task, and in spite some preliminary accumulation of experience, theoretical framework and sound practical guidelines are non-existent or just emerging. Nevertheless, the subject is gaining increasing attention, and will continue to do so as complex databases become more common.

The penalty for restructuring, beyond the obvious direct effort in its design and implementation, is twofold. First, it may involve a relatively long period of lower availability of the DB, and second, which is virtually neglected, is that it reduces substantially the accessibility of past data, for clear technical as well as behavioral reasons. The Modal Data Management practically dissolves the problem by internalizing the change itself and maintaining multiple schemata, each of them associated with certain time period and serves as basis for access to data pertaining to that period. Note, however, that the continuity in the factual basis is preserved through the backward pointing included in each transaction.

By lowering the cost of getting results equivalent to traditional restructuring, we clearly increase the incentive to keep the functional capabilities of the system closer to their "ideal" position. In a sense we are managing the DB Definition along the MDM conventions! This concept can be extended to include software modules, namely both schemata and modules will never be deleted, and rather serve as a "Time Dependent Access Method". This notion integrates into the system possible solution to the issues that are motivating the emerging concept of Software Configuration Management [17].

## 6.0 CONCLUSION

Modal Data Management provides an alternative to the conventional notions of DBMS, and enhances it substantially in terms of usability.

This is not to say that existing systems do not handle time. Conventional systems do deal, in certain cases, with time oriented data (e.g., salary payments in the current budget year, sales summary for the current year) and in some cases they do it even quite extensively (for instance, data concerning securities in Stock Exchange DB). However, this is usually the result of a primary design decision, which means complete concentration on the time aspect, at the expense of different views of the data. For example, in typical Stock Exchange DBMS, process oriented questions, e.g, "what has been the volume of trade by agent XYZ" are answerable only by using a separate DB. Another phenomenon in the time handling within the context of conventional DBs is the time-discontinuities in the availability of temporal data due to a periodic "clean up" (e.g. last year salaries are being removed from the DB on December 31st, while actually inquiries to that data is very likely to occur even later).

The Modal Data Management preserves the basic, implicit nature of time, i.e, even though we may not be fully aware of it, we are constantly moving in time. As in real life, events recorded in the MDM's DB are necessarily anchored in a temporal context, apart from other semantic context.

Modal Data Management also smoothes the transition from one "world view" to the next, recognizing that we do not live in a pre-ordained world. In so doing it also mitigates some of the major problems that Systems Analysis has in dealing with dynamic processes. Given the highly dynamic nature of office processes, our current attention is focused on building an MDM system for OFFICE AUTOMATION processes. This must deal not only with structured fields but also with text.

References


1. Anderson, T.L. "Database Semantic of Time", Ph.D Dissertation, Computer Science Dept., U. of Washington, 1981.

2. Ariav, G. "Graphic Output for 'Non-Graphical' Databases", in preparation.

3. Bubenko, J.A. "Information Modeling in the Context of System Development", Information Processing 80, S.H. Lavington (Ed.), North-Holland/IFIP, 1980, 395-411.

4. Buneman, O.P., and Morgan, H.L. "Implementing Alerting Techniques in Database Systems", Proceedings of the IEEE Computer Society's First International Computer & Applications Conference, November 1977.

5. Copeland, G. "What if Mass Storage Were Free?", Proceedings of the 1980 Workshop on Computer Architecture for Non-Numeric Processing, ACM and IEEE, Pacific Grove, CA, 1980, 1-7.

6. Donovan, J.J "Database System Approach to Management Decision Support", ACM Trans on Databse Syst., 1, 4 (December 1976), 344-369.

7. Emery, J.C. Personal Communication.

8. Fraser, J.T., (Ed.) The Voices of Time, George Braziller, Inc., New York, 1966.

9. Gallaire, H., and Minker, J., (Eds.) Logic and Databases, Plenum Press, New York, 1978.

10. Herrot, C.F. "Spatial Management of Data", ACM Trans on Database Syst., 5, 4 (December 1980), 493-513.

11. Kahn, K.M. "Mechanization of Temporal Knowledge", Massachusetts Institute of Technology Technical Report MAC-TR-155, MIT, Cambridge, Mass., 1975.

12. Keen, P.G.W., and Scott Morton, M.S. Decision Support Systems: An Organizational Perspective, Addison-Wesley, Reading, Mass., 1978.

13. Kimball, K.A., and Morgan, H.L. "The DATA System", Decision Sciences Working Paper 78-04-03, Dept. of Decision Sciences, U. of P., Philadelphia, PA, 1978.

14. SEED User Manual (B.11), International Data Base Systems, Inc., Philadelphia, PA, August 1980.

15. Severance, D.G., and Lohman, G.M. "Differential Files: Their Application to the Maintenance of Large Databases", ACM Trans on Database Syst., 1, 3 (September 1976), 256-267.

16. Sherover, C.M. The Human Experience of Time, New York University Press, New York, 1975.

17. Sibley, E. H., Scallan, P. G., and Clemons, E. K. "The Software Configuration Management Database", NCC, AFIPS Press, 1981, 249 - 255.

18. Sockut, G. H., and Goldberg, R. P. "Database Reorganization - Principles and Practice", ACM Computing Surveys, 11, 4 (December 1979), 371-395.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

# REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER: 81-05-06 | 2. GOVT ACCESSION NO.: AD-A103 308 |
| 3. RECIPIENT'S CATALOG NUMBER: | |
| 4. TITLE (and Subtitle): MDM: Handling The Time Dimension in Generalized DBMS | 5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report |
| | 6. PERFORMING ORG. REPORT NUMBER: 81-05-06 |
| 7. AUTHOR(s): Gad ARIAV and Howard Lee MORGAN | 8. CONTRACT OR GRANT NUMBER(s): MDA903-80-C-0093 (and N00014-75-C-0462) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Philadelphia, PA 19104 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: PH-213 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Department of the Navy, DARPA | 12. REPORT DATE: May 81 |
| | 13. NUMBER OF PAGES: 34 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | 15. SECURITY CLASS. (of this report): unclassified |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Time in Information Systems. Data Base Management Systems (DBMS). Temporal Data Storage

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The sense of time is implicit in almost all human activity, yet it is rarely reflected in the computer database views of these activities. This paper offers a method of dealing with time, modal storage and retrieval, and describes formal and practical realizations of the concept.

The conceptual framework of Modal Data Management reflects three years of experience with the DATA (Dynamic Alerting Transaction Analysis) system (13)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

(continued on reverse)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(20) cont'ed

in which a time oriented DBMS has been conceived, designed and actually implemented.

The paper first anchors the Modal Data Management concept in the context of the relevant research in Information Modelling and storage technologies, folllowed by a discussion of the architectural and functional attributes of the DATA system. The major lessons from the DATA experience are then presented, paying special emphasis to their impact on the design of future versions of the system.

The concluding comments deal with specific implications of the Modal Data Management in the domains of DBMS usability and Software Engineering.


DAT
ILM